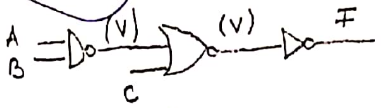
4. SEQUENCIAL STATES

1. VARIABLE -----------------------------------

```
process (A, B, C)
    variable V: Std_logic;              -- local !

begin
    V := A nand B;
    V := V nor C;
    F <= not V;
end process;
```

– doar în proces

V- creează 2 nivele.



→ ∄ delay

Comparație

	Signal	Variable
1)	● Arh – Begin globale	● Proc → Begin locale
2)		
3)	are delay	Nu delay

= procesele comunică prin signal

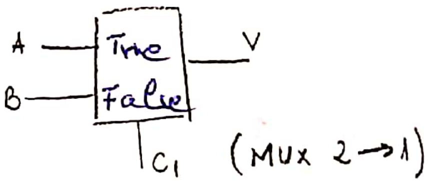
2. SIGNAL / VARIABLE --------------------

```
architecture ...
    signal F: Std_logic;                -- global !

begin
    process (A, B, C)
        variable V: Std_logic;          -- local !
    begin
        V := A nand B;
        V := V nor C;
        F <= not V;
    end process;

    process (F)
    begin
    ...
    end process;
    ...
```

3. IF / ELSE → MUX ---------------------------

```
    if C1 = '0' then
        V := A;
    else
        V := B;
    end if;
```

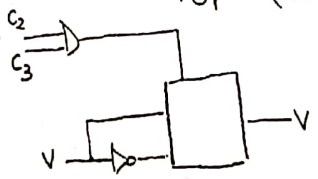
MUX



4. ---

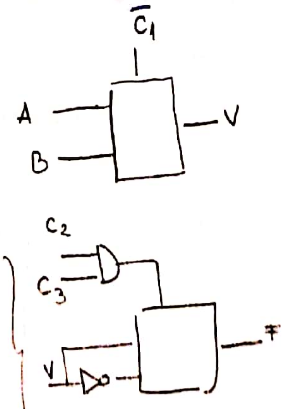
```
    if C2 = '1' and C3 = '1' then
        V := not V;
    end if;
```

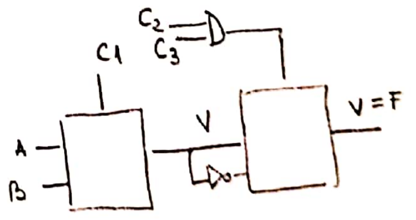
→ fără else



5. ---

```
process (C1, C2, C3, A, B)
    variable V: Std_logic;
begin
    if C1 = '0' then
        V := A;
    else
        V := B;
    end if;

    if C2 = '1' and C3 = '1' then
        V := not V;
    end if;

    F <= V;
end process;
```



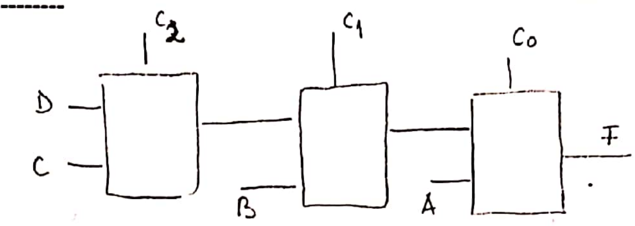


↓ cod

implem. cu MUX

6. NESTED IF ----------------------------------

```
process (C0, C1, C2, A, B, C, D)

begin
    if C0 = '1' then
        F <= A;
    else
        if C1 = '1' then
            F <= B;
        else
            if C2 = '1' then
                F <= C;
            else
                F <= D;
            end if;
        end if;
    end if;
end process;
```



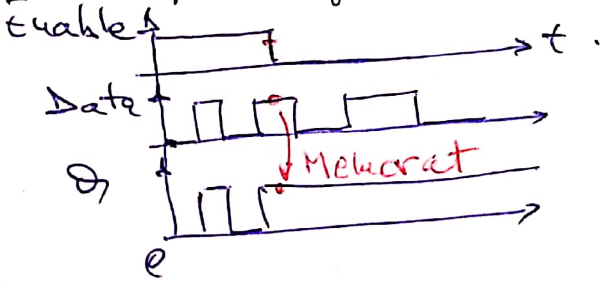
iF înlănțuite

7. ELSIF --

```
process (C0, C1, C2, A, B, C, D)

begin
    if C0 = '1' then
        F <= A;
    elsif C1 = '1' then
        F <= B;
    elsif C2 = '1' then
        F <= C;
    else
        F <= D;
    end if;
end process;
```

Ac. funcție, alternativă, scrisă condensată. (ca' la ex. 6)

Permite teste multiple cascadate : se exec. doar ramura coresp. cond. logice adevărate.



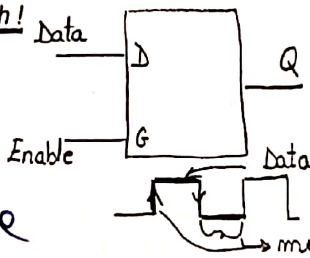
8. INCOMPLET ASIGN ----------------------

```
process (Enable, Data)

begin
    if Enable = '1' then
        Q <= Data;
    end if;
end process;
```

fără else

=> circ. cu memorie

-- *transparent latch !*



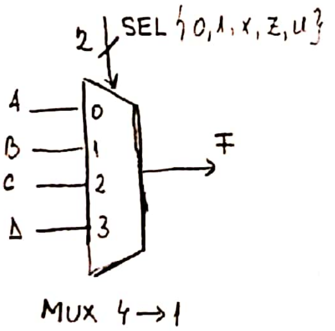
Data = Q (avem fir între ele)

memorează Data.

9. CASE --

```
    case SEL is
        when "00" =>
            F <= A;
        when "01" =>
            F <= B;
        when "10" =>
            F <= C;
        when "11" =>
            F <= D;
        when others =>
            F <= 'X';
    end case;
```

-- *nu elemente comune !*



```
10. OR  COND ---------------------------------
    case ADDRESS is
        when 16 | 20 | 24 | 28 =>
            A <= '1';
            B <= '1';
        when others =>
    end case;
```

sau (se pot acoperi m.m. cazuri într-o sg. ramură)

| ← SAU

} → stări secv. se separă prin ;

nu se modifică nimic
(se memorează vechea val. a lui A, B)

```
11. DOMAIN COND ----------------------------
    case ADDRESS is
        when 0 to 7 =>
            A <= '1';
        when 8 to 15 =>
            B <= '1';
        when 16 | 20 | 24 | 28 =>
            A <= '1';
            B <= '1';
        when others =>
            null;
    end case;
```

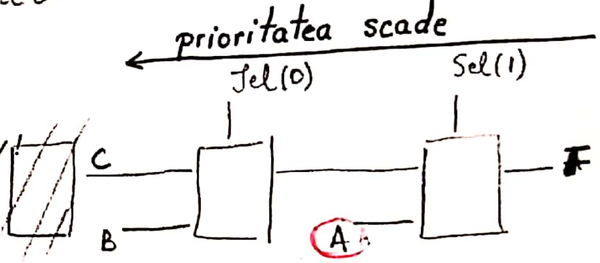
(∀) adresa cuprinsă între 0 și 7 (adică 8 adrese) acoperă un întreg subdomeniu.

La m.m. WHEN-uri e interzis să avem elem. comune.

no action (~~nu se schimbă~~ nimic)

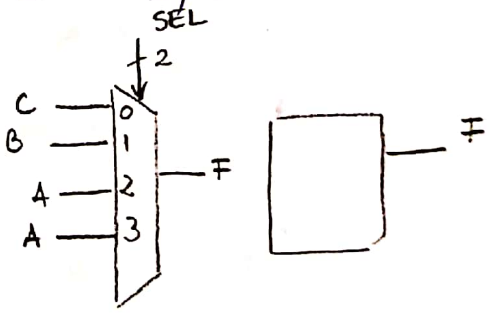
→ ✓ ~~A, B sunt puse pe 0~~

```
12. COMPARISON IF- CASE---------------        -- IF priority !
   if SEL(1) = '1' then
        F <= A;
   elsif SEL(0) = '1' then
        F <= B;
   else
        F <= C;
   end if;
```



nivele log = nr. if → lent
n x (1-2 ns)

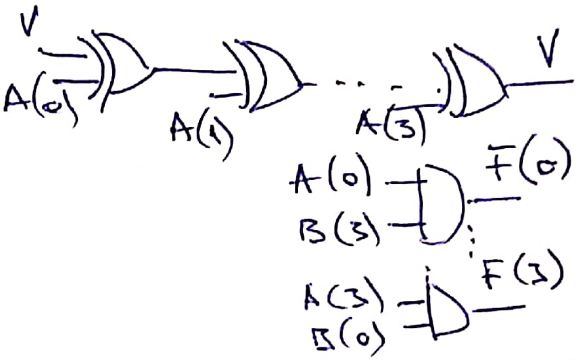
```
13. ----------------------------------------        -- CASE paralel test !
   case SEL is
        when "10" =>
            F <= A;
        when "11" =>
            F <= A;
        when "01" =>
            F <= B;
        when others =>
            F <= C;
   end case;
```



```
14. ----------------------------------------        -- No FPGA !
   if SEL = "00" then
        F <= A;
   elsif SEL = "01" then
        F <= B;
   elsif SEL = "10" then
        F <= C;
   else
        F <= D;
   end if;
```

if	case
∃ priorit.	∄ prior.
Lent - n x nivele	Rapid - 1 nivel

```
15. -------------------------------------------------
    case SEL is
        when "00" =>
            F <= A;
        when "01" =>
            F <= B;
        when "10" =>
            F <= C;
        when others =>
            F <= D;
    end case;
```



(1)

```
16. FOR LOOPS --------------------------------    -- [Lab:] for Loop_par in Range loop
    for I in 0 to 3 loop
        F(I) <= A(I) and B(3-I);
        V := V xor A(I);
    end loop;                                     -- end loop [Lab];
-------------------------------------------------

    for I in 3 downto 0 loop                      -- descendent range
        F(I) <= A(I) and B(3-I);
        V := V xor A(I);
    end loop;
```

local

⇒ n = 4 copii corp for

```
17. ---------------------------------------------
process (A, B)
    variable I: Std_logic;
...
begin

    for I in 0 to 3 loop                          -- I local !
        F(I) <= A(I) and B(3-I);
        V := V xor A(I);
    end loop;

    I := not I;
end process;
```

→ se inițializează cu 0

→ o altă variabilă I locală.

contor for

→ i se inițializează cu 1.

```
18. MULTIPLE COPY ----------------------
process (A, B)
    variable V: Std_logic;

begin
    V := '0';

    for I in 0 to 3 loop
        F(I) <= A(I) and B(3-I);
        V := V xor A(I);
    end loop;

    G <= V;
end process;
```

→ signal declarat în arhitectură, pt. a fi vizibil rezultatul înafara procesului;

```
end process;
```

2

```
19 LOOP ---------------------------------------
   loop
      ...
      exit;
      ...
      exit when CONDITION;
      ...
   end loop;
```

prezintă facilități de ieșire din LOOP

```
20. EXIT Label WHEN ----------------------
   L1: for I in 0 to 7 loop
      L2: for J in 0 to 7 loop
         C := C + 1;
         exit L2 when A(J) = B(I);      -- label shows which loop exit !
         exit L1 when B(C) = 'U';
      end loop L2;
   end loop L1;
```

2 for imbricate

obligatoriu utilizarea etichetelor (L1, L2)

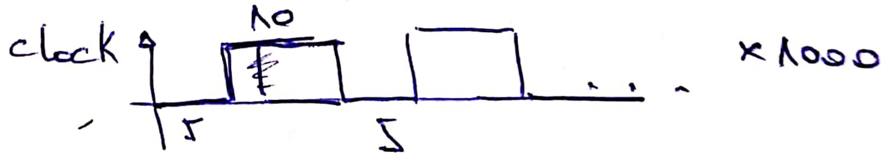
nu sunt obligatorii.

```
21. ---------------------------------------------
   Main: for I in 0 to 15 loop
      ...
      next Main when Reset = '0';      -- jmp to begin !
      ...
   end loop Main;
```

bucla nu se mai execută până la sfârșit, dacă Reset='0' se sare la începutul lui FOR și se incrementează contorul.

```
22. TEST BENCH CLOCK -------------------
ClockGenerator_1: process

begin
   for I in 1 to 1000 loop                -- eqv: while NOW < 15 US
      Clock <= '0';
      wait for 5 NS;
      Clock <= '1';
      wait for 10 NS;
   end loop;

   wait;
end process ClockGenerator_1;
```

-generator de CLK pt. TESTBENCH, dar cu nr. limitat de tacte.



≡ 23

```
23. ---------------------------------------------
ClockGenerator_3: process

begin
   loop
      Clock <= '0';
      wait for 5 NS;
      Clock <= '1';
      wait for 10 NS;
      exit when NOW >= 15 US;
```

NOW - ceas implicit = timp simul

```
      end loop;

      wait;
end process ClockGenerator_3;
```

24. ---

```
process (A, B, C, D)                          -- priority coder ?
      variable V: Std_logic_vector(3 downto 0);

begin
      V(3) := A;
      V(2) := B;
      V(1) := C;
      V(0) := D;
      F <= 0;

      for I in 0 to 3 loop
            if V(I) = '1' then
                  F <= I;
                  exit;
            end if;
      end loop;

end process;
```

(U,X)

codor prioritate

→ prioritate scade

	D	C	B	A
i	0	1	2	3

→ F - contine indicele variabilei ~~cele~~ cu cea mai mare priorit

25. ---

```
architecture V2 of AOI is
      signal AB, CD, O: STD_LOGIC;

begin
      AB <= A and B after 2 NS;               -- event asign !
      CD <= C and D after 2 NS;
      O <= AB or CD after 2 NS;
      F <= not O after 1 NS;
end V2;
```

✓ ∄ proces

26. ---

```
process (A, B)

begin
      AB <= A and B after 2 NS;
end process;
```

Scanned with CamScanner